

## 11、串口通讯-串口控制 LED

### 1.实验目的

- 1)、通过实验掌握 CC2530 芯片串口配置与使用
- 2)、接收串口发送过来的数据，通过数据内容分析控制 LED

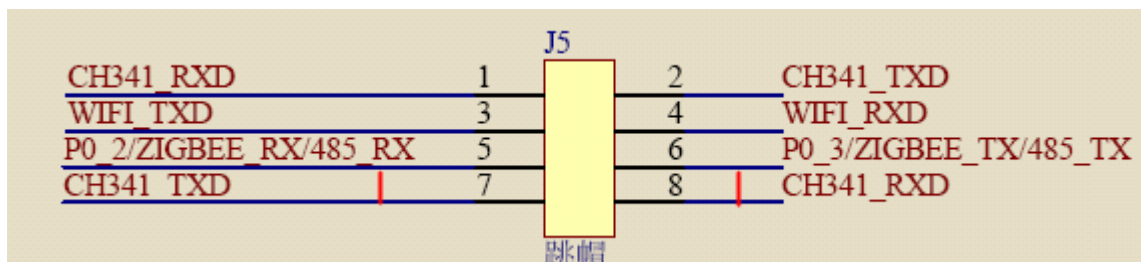
注：嵌入式开发中，当程序能跑起来后，串口是第一个要跑起来的设备，所有的工作状态，交互信息都会从串口输出。

### 2.实验设备

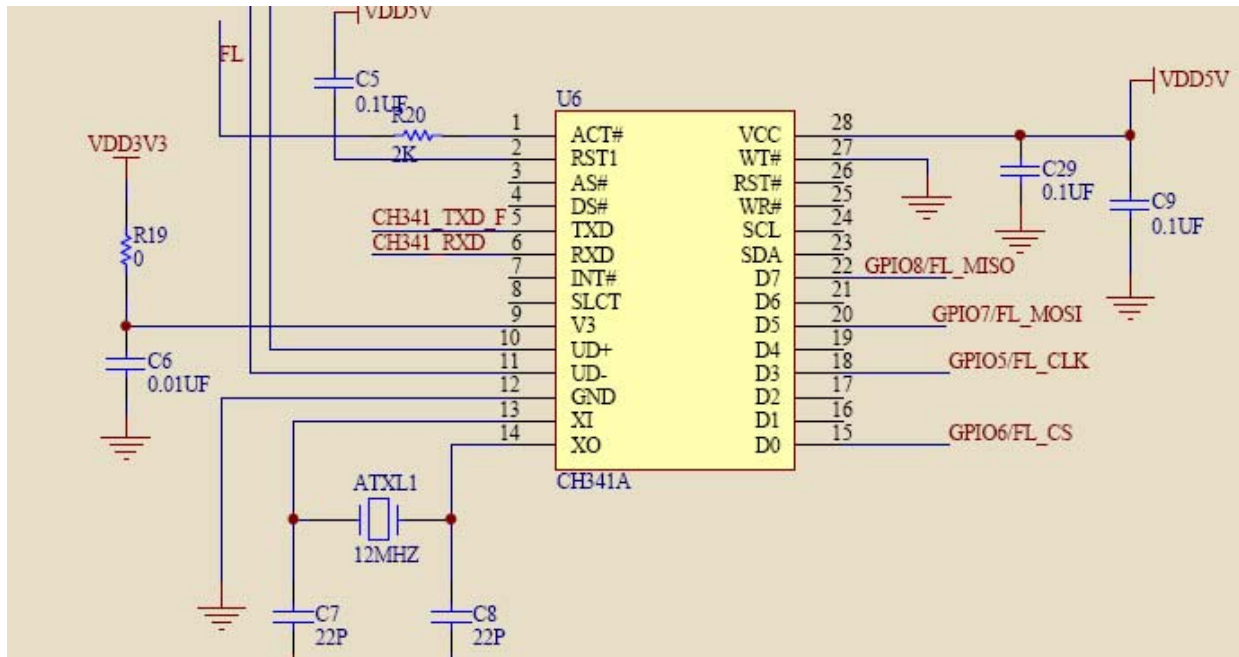
硬件：PC 机一台 ZB 网关（底板、核心板、仿真器、USB 线）一套

软件：2000/XP/win7 系统，IAR 8.10 集成开发环境、串口助手

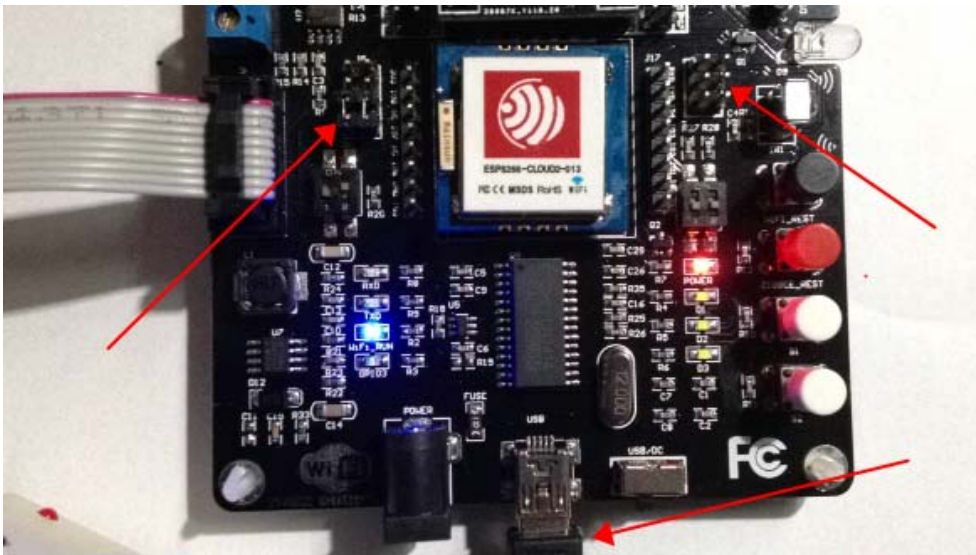
### 3.相关电路图



如上图 J5 这样接，表明 PC 通过 USB 线[板子自带 USB 转串口芯片 CH341]直接接入 ZB 模块，进行对 ZB 的调试使用。如果 3-5 4-6 则表明 ZIGBEE 通过串口控制 wifi 模块。如果 1-3 2-4 则表明 WIFI 模块接入到 PC



如下图就是我们板子实际的短路帽情况：



硬件我们准备好了。我们将板和电脑连接。这个时候需要安装驱动。具体安装方法很简单，下载驱动精灵自动安装。不要自己安装，自己安装容易把驱动搞乱。驱动精灵网址：

<http://pan.baidu.com/s/14fUyU>



P0\_2、P0\_3 配置为外设功能时：P0\_2 为 RX，P0\_3 为 TX。USART0 和 USART1 是串行通信接口，它们能够分别运行于异步 UART 模式或者同步 SPI 模式。两个 USART 具有同样的功能，可以设置在单独的 I/O 引脚。此种串口设计是没有流控功能的。

#### 4.相关寄存器

相关寄存器 UxCSR、UxCSR、UxGCR、UxBUF、UxBAUD、CLKCONCMD、CLKCONSTA 如

寄存 器	位	描述
UOCSR (0x86) -USART0 控制 和状态	Bit[7] MODE	USART 模式选择 0: SPI 模式 1: UART 模式
	Bit[6] RE	UART 接收器使能 0: 禁用接收器 1: 接收器使能
	Bit[5] SLAVE	SP 主或者从模式选择 0: SPI 主模式 1: SPI 从模式
	Bit[4] FE	UART 帧错误状态 0: 无帧错误检测 1: 字节收到不正确停止位级别
	Bit[3] ERR	UART 奇偶错误状态 0: 无奇偶错误检测 1: 字节收到奇偶错误
	Bit[2] RX_BYTE	接收字节状态 0: 没有收到字节 1: 准备好接收字节
	Bit[1] TX_BYTE	传送字节状态 0 字节没有被传送 1 写到数据缓存寄存器的最后字节被传送
	Bit[0] ACTIVE	USART 传送/接收主动状态、在 SPI 从模式下 该位等于从模式选择 0: USART 空闲 1: 在传送或者接收模式 USART 忙碌
UOOCR (0xC5) -USART0 通用 控制	Bit[7] CPOL	SPI 的时钟极性 0: 负时钟极性 1: 正时钟极性
	Bit[6] CPHA	SPI 时钟相位 0: 当 SCK 从 CPOL 倒置到 CPOL 时数据输出 到 MOSI, 并且当 SCK 从 CPOL 倒置到 CPOL 时 数据输入抽样到 MISO。 1: 当 SCK 从 CPOL 倒置到 CPOL 时数据输出 到 MOSI, 并且当 SCK 从 CPOL 倒置到 CPOL 时 数据输入抽样到 MISO

	Bit[5] ORDER	传送位顺序 0: LSB 先传送 1: MSB 先传送
	Bit[4:0] BAUD_E	波特率指数值。BAUD_E 和 BAUD_M 决定了 UART 波特率和 SPI 的主 SCK 时钟频率
U0BAUD (0xC2) - USART 0 波特率控制	BAUD_M[7:0]	波特率小数部分的值。BAUD_E 和 BAUD_M 决定了 UART 的波特率和 SPI 的主 SCK 时钟频率
U0DBUF		USART 0 接收/发送数据缓存
UTX0IF(发送中断标志)	IRCON2 Bit1	USART 0 TX 中断标志 0: 无中断未决 1: 中断未决
CLKCONCMD 时钟控制命令	Bit[7] OSC32K	32 kHz 时钟振荡器选择 0 : 32 kHz XOSC      1 : 32 kHz RCOSC
	Bit[6] OSC	系统时钟源选择 0 : 32 MHz XOSC1 : 16 MHz RCOSC
	Bit[5:3] TICKSPD	定时器标记输出设置 000 : 32 MHz    001 : 16 MHz    010 : 8 MHz 011 : 4 MHz    100 : 2 MHz    101 : 1 MHz 110 : 500 kHz   111 : 250 kHz
	Bit[2:0] CLKSPD	时钟速度 000 : 32 MHz    001 : 16 MHz    010 : 8 MHz 011 : 4 MHz    100 : 2 MHz    101 : 1 MHz 110 : 500 kHz   111 : 250 kHz
CLKCONSTA		CLKCONSTA 寄存器是一个只读寄存器, 用来获得当前时钟状态

由寄存器UxBAUD.BAUD\_M[7:0]和UxGCR.BAUD\_E[4:0]定义波特率。该波特率用于UART 传送，也用于SPI 传送的串行时钟速率。波特率由下式给出：

$$\text{波特率} = \frac{(256 + \text{BAUD\_M}) * 2^{\text{BAUD\_E}}}{2^{28}} * F$$

F 是系统时钟频率，等于 16 MHz RCOSC 或者 32 MHz XOSC。

### 32 MHz 系统时钟常用的波特率设置

波特率(bps)	UxBAUD. BAUD_M	UxGCR. BAUD_E	误差(%)
2400	59	6	0.14
4800	59	7	0.14
9600	59	8	0.14
14400	216	8	0.03
19200	59	9	0.14
28800	216	9	0.03
38400	59	10	0.14
57600	216	10	0.03
76800	59	11	0.14
115200	216	11	0.03
230400	216	12	0.03

### CC2530 配置串口的一般步骤：

- 1、配置 IO，使用外部设备功能。此处配置 P0\_2 和 P0\_3 用作串口 UART0
- 2、配置相应串口的控制和状态寄存器。
- 3、配置串口工作的波特率。

由于此实验增加了串口接收功能，寄存器有所改变(红色部分)，具体配置如下：

PERCFG = 0x00; //位置1 P0 口

P0SEL = 0x0c; //P0\_2,P0\_3用作串口（外部设备功能）

P2DIR &= ~0XC0; //P0优先作为UART0

U0CSR |= 0x80; //设置为UART方式

U0GCR |= 11;

U0BAUD |= 216; //波特率设为115200 根据上面表中获得的数据

UTX0IF = 0; //UART0 TX 中断标志初始置位0

```
U0CSR |= 0x40; //允许接收
```

```
IEN0 |= 0x84; //开总中断允许接收中断
```

## 5.源码分析

```
#include <iocc2530.h>
```

```
#include <string.h>
```

```
#define uint unsigned int
```

```
#define uchar unsigned char
```

```
//定义控制灯的端口
```

```
#define LED1 P1_0
```

```
#define LED2 P1_1
```

```
#define LED3 P0_4
```

```
void initUART0(void);
```

```
uchar Recdata[3]="000";
```

```
uchar RXTXflag = 1;
```

```
uchar temp;
```

```
uint datanumber = 0;
```

```
uint stringlen;
```

```
/*  
*****  
串口 ( UART0)初始化函数 :  
*****  
*/
```

```
串口 ( UART0)初始化函数 :
```

```
*****  
*/
```

```
void initUART0(void)
```

```
{
```

```
CLKCONCMD &= ~0x40;
```

```
//设置系统时钟源为 32MHZ 晶振
```

```
while(CLKCONSTA & 0x40);
```

```
//等待晶振稳定
```

```
CLKCONCMD &= ~0x47;
```

```
//设置系统主时钟频率为 32MHZ
```

```

PERCFG = 0x00;           //位置 1 P0 口
POSEL = 0x0c;           //P0 用作串口
P2DIR &= ~0XC0;         //P0 优先作为 UART0

U0CSR |= 0x80;           //串口设置为 UART 方式
U0GCR |= 11;
U0BAUD |= 216;           //波特率设为 115200
UTX0IF = 1;             //UART0 TX 中断标志初始置位 1

U0CSR |= 0x40;           //允许接收
IEN0 |= 0x84;           //开总中断，接收中断
}

/*****
初始化 LED IO 口
*****/

void Init_LED_IO(void)
{
    P1DIR = 0x03;         //P10 P11 为输出
    P0DIR = 0x10;         //P14 为输出
    LED1 = 1;
    LED3 = 1;
    LED2 = 1;             //灭 LED
}

/*****
主函数
*****/

void main(void)
{
    uchar i;
    Init_LED_IO();

```



```
initUART0();
while(1)
{
if(RXTXflag == 1)           //接收状态
{
    if( temp != 0)
    {
        if((temp!='#')&&(datanumber<3))
        {
            // # '被定义为结束字符'
            //最多能接收 3 个字符
            Recdata[datanumber++] = temp;
        }
        else
        {
            RXTXflag = 3;           //进入改变小灯的程序
        }
        if(datanumber == 3)RXTXflag = 3;
        temp = 0;
    }
}
if(RXTXflag == 3)
{
    if(Recdata[0]=='R')
    {
        if(Recdata[1]=='0')LED1 = 1;    // R0# 关 D1
        else
            LED1 = 0;                   // R1# 开 D1
    }
    if(Recdata[0]=='Y')
    {
        if(Recdata[1]=='0')LED3 = 1;    // Y0# 关 D3
    }
}
```

```
else
    LED3 = 0;           // Y1# 开 D3
}
if(Recdata[0]=='G')
{
    if(Recdata[1]=='0')LED2 = 1;    // G0# 关 D2
    else
        LED2 = 0;           // G1# 开 D2
}
if(Recdata[0]=='A')
{
    if(Recdata[1]=='0')
    {
        LED1 = 1;
        LED3 = 1;
        LED2 = 1;           // A0# 关所有 LED
    }
    else
    {
        LED1 = 0;
        LED3 = 0;
        LED2 = 0;           // A1# 开所有 LED
    }
}
RXTXflag = 1;
for(i=0;i<3;i++)Recdata[i]=' '; //清除刚才的命令
datanumber = 0;           //指针归 0
}
}
}

/*****
```

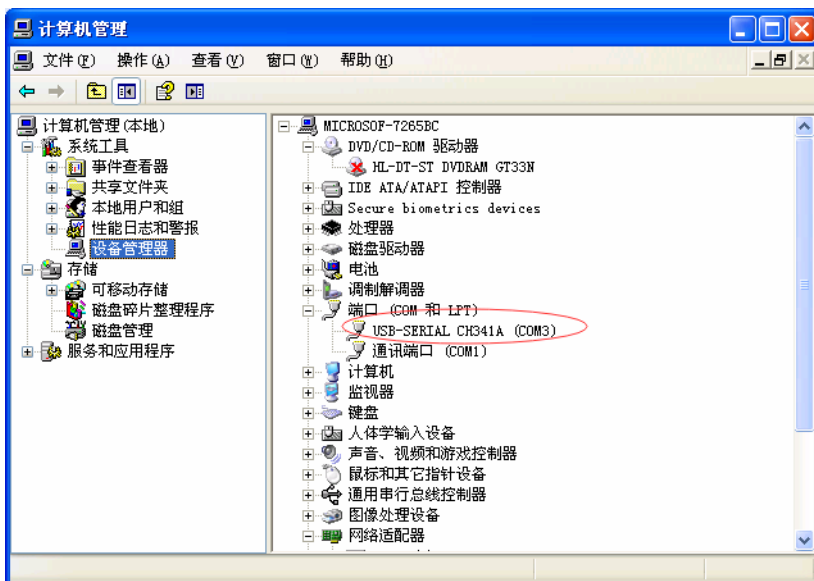
串口接收一个字符:一旦有数据从串口传至 CC2530,则进入中断,将接收到的数据赋值给变量 temp.

\*\*\*\*\*/

```
#pragma vector = URX0_VECTOR
__interrupt void UART0_ISR(void)
{
    URX0IF = 0;           //清中断标志
    temp = U0DBUF;
}
```

## 6、实验现象

COM3 是我的 USB 转串口在电脑上生成的,查看方法“我的电脑” - > “设备管理器”, 如图:



## 6、实验现象

串口字符控制灯状态表

串口发送字符 灯状态

R0#	D1 关
R1#	D1 开
G0 #	D2 关
G1 #	D2 开
Y0#	D3 关
Y1#	D3 开
A0 #	关D1 , D2 , D3
A1 #	开D1 , D2 , D3

如下图这样发



